THE ACEDB GENOME DATABASE

Richard Durbin [1] and Jean Thierry-Mieg [2]

[1] MRC Laboratory of Molecular Biology, Hills Road,
Cambridge CB2 2QH, UK (email: rd@mrc-lmb.cam.ac.uk).

[2] CNRS Physique Mathematique and CRBM, BP 5051
34033 Montpellier, France (email: mieg@kaa.cnrs-mop.fr)

INTRODUCTION

Systematic genome mapping and sequencing projects are generating resources that will permanently change the practice of molecular biology. To maximise their effect, we have to make the information available to the scientific community in as useful a form as possible. It has been said that the sheer quantity of genomic information that we are just now beginning to gather will cause problems for any database system that must store it. That is not in itself strictly true; in fact the current total of genome mapping and sequence data, for all organisms combined, would sit comfortably in a one gigabyte disk, which is small for a workstation, and even conceivable for a PC. Furthermore, although the amount of genome data being collected is undergoing exponential growth, so is the capacity of computer storage systems, with an even shorter doubling time, so the issue of raw storage capacity is becoming progressively easier.

However, in another sense, the fear of data overload has some justification. This is because the issue is not just one of simple storage, but of integrating the new systematic data with all our accumulated experimental knowledge from classical and molecular genetics, so as to be able to select what is relevant for each scientific question. Until recently, with the results disseminated by standard scientific publication and discussion, this process of integration has taken place in the minds of the researchers. Even if all details could not be followed, the salient facts involved with some specialised function could be managed. It is this storage medium, the human brain, that is incapable of handling all the genomic data, not the computer.

Clearly what is required is a database system that, in addition to storing the results of large scale sequencing and mapping projects, allows all sorts of experimental genetic data to be maintained and linked to the maps and sequences in as flexible a way as possible. Since this is a new type of system, it seems very desirable to have a database whose structure can evolve as experience is gained. However this is in general very difficult with existing database systems, both relational, such as Sybase and Oracle, and object-oriented, such as Object Store.

We were faced with these issues three years ago, when starting a pilot project for obtaining the complete genomic sequence of the nematode C. elegans. There were dual needs: first for a system in which to maintain data for internal purposes, and second for one in which to make it public. We wanted to build on previous experience gained while building the physical clone map for C. elegans, which had been done using the program CONTIG9 [1]. An

Computational Methods In Genome Research,
Edited by S. Suhai, Plenum Press, New York, 1994

adapted form of this program, called PMAP, was made publicly available to the C. elegans research community, together with regularly updated copies of the in house data. This rapid and complete access to the map, even in incomplete form, proved to be extremely popular and successful, soon becoming a crucial resource when cloning worm genes. It therefore seemed sensible to extend the same approach, and develop a single database to hold sequence, physical and genetic map, and references, that we could use in house, and that we could distribute in read-only form freely within the worm community.

This led directly to the database program ACEDB, which is described in this chapter. Rather than being limited to the specific data that we could envisage when we started, we decided to write a general database management system that would allow easy and frequent extension and adaptation of the database schema as the project developed. For this reason, it has been comparatively easy to adapt ACEDB to be used by other genome projects working with other organisms. At the time of writing (March, 1993) there are public databases for the model plant Arabidopsis thaliana, and the mycobacteria M. leprae and M. tuberculosis, which are the pathogens for leprosy and tuberculosis. Several other databases for public distribution are under development. ACEDB is also being used internally at several sites, for example for storage of physical mapping results from human and Drosophila projects. Finally, it is being used as one of the core pieces of software in the IGD (Integrated Genomic Datatape) project (European Data Recource for Human Genom Research, Heidelberg), which plans to bring together all public human genome data in an integrated genome database. ACEDB is both being used as the primary graphics front end of IGD, and as one of the alternative back-end data storage systems.

OVERVIEW OF FEATURES

In this section we will give a brief overview of ACEDB as a biological user sees it. Overall the program is very graphical. It works using a windowing system, and presents data in different types of window according to the different types of map. The maps and other windows are linked in a hypertext fashion, so that clicking on an object will display further information about that object in the appropriate sort of window. For example, clicking on a chromosome displays its genetic map; clicking on a gene in the genetic map displays text information about the gene's phenotype, references etc; clicking on a clone displays the physical map around it; clicking on a sequence starts the sequence display facility.

The internal structures of the system, which are more general, and which contain some of the more novel features, will be described in later sections. There are interactive tools for many of these more general features available to the user as part of the graphical interface, but discussion of these will be delayed until later. In this section we will just briefly describe the windows used to display the different classes of object in the database.

MAIN CONTROL WINDOW

There is one window that is always present while running ACEDB: the main control window. This contains a list of the different available classes of objects, such as Clones, Sequences, Genes, Papers, Authors etc. To use the program in its most simple fashion the user selects a class with the mouse and types the name of the object in the yellow text entry box, then hits the return key, at which point the object will be displayed in another window. If a name template is given using wildcard characters (such as '*') then a list of all possible matching names is given, from which the user can select. There is also a menu accessible with the right mouse button that provides access to more complex features, such as the query system discussed later.

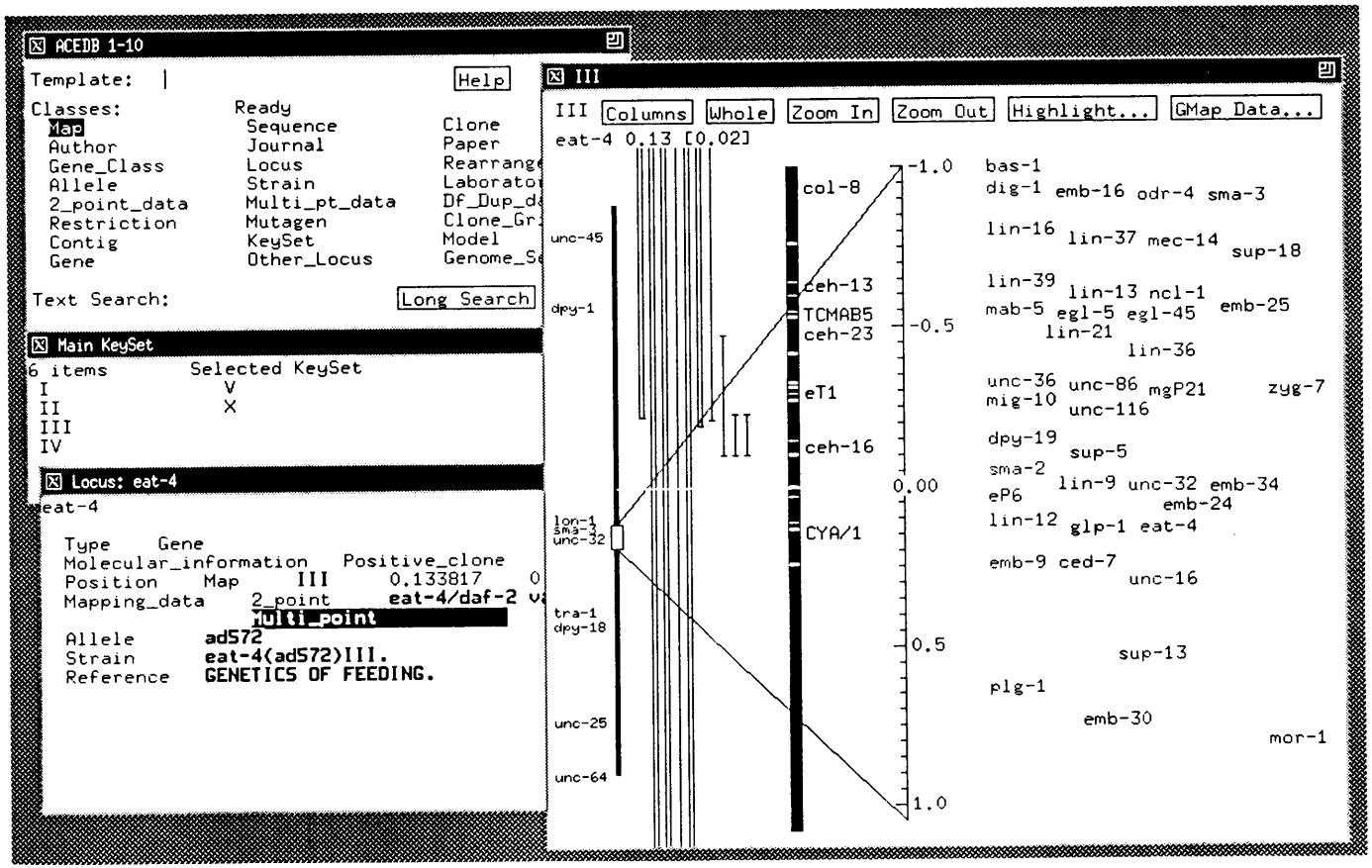

Figure 1. Main window, Genetic map, and text windowof one gene

GENETIC MAP

The genetic map display gives access to genes and rearrangements on the scale of a chromosome. In the C. elegans map the units used are centimorgans. The map is displayed vertically, as are many ACEDB maps. On the left side, acting like an annotated scrollbar, is an image of the whole chromosome with a green cursor region indicating the currently displayed region. This area is zoomed up to fill the rest of the display, which consists of various zones, including from left to right: space for chromosomal rearrangements (deficiencies and duplications), an indication of physically mapped regions, a scale bar, and the genes themselves. The display can be easily scrolled up and down, and zoomed in and out using either the scrollbar cursor, or buttons that can be pressed with with the mouse. There are also buttons to allow genetic mapping data to be displayed graphically. Recently we have extended the mapping data package to also allow calculations on mapping positions to be made from the map data.

If the user double clicks on any item a new window pops up with text information about the object. This text information is layed out hierarchically, in what is called a tree structure. The section below on "Organisation of data" describes further this tree structure, which in fact is the primary way of storing information in ACEDB. The maps are merely derived from the data stored in tree form with each object.

PHYSICAL MAP

This is the primary display for looking at the positions of clones within clone contigs. It is (currently) a horizontal map, based very much on the map display of CONTIG9 [1]. Again at the bottom is an annotated scrollbar showing a wider region of the contig. At the top are a number of sections showing different types of clone, i.e. probes, YACS, cosmids. Under these are spaces for genes that have been attached to the map by localising them onto a clone, and remarks, which can be freely attached to any clone. Also in this region is a green horizontal band linking back to the genetic map. Again, whenever an item is selected, all the related items are also lit up; e.g. selecting a gene might show the clone that contains it, and any remarks attached to the clone. If the item is double clicked, then text information about the item is displayed.

HYBRIDISATION GRID

This window provides access to one of the forms of raw data used in building the physical map, which is hybridisation of probes to a grid of clones arrayed on nylon filter. The grid is displayed schematically, and the hybridisation pattern can be entered by clicking on the appropriate squares. Once this is done, the inferred loci on the physical map are determined automatically. There are also facilities for comparing one pattern with another, and for displaying the results of both real and hypothetical pooled probings. This tool was used to position 1100 nematode cDNA's on the C. elegans physical map, and is also used by a human mapping project for data acquisition.

DNA SEQUENCE

The sequence display function in ACEDB is particularly flexible. As well as being able to display the actual nucleotide sequence and protein translations of it in all frames, there are a wide range of different schematic display options available. These allow features to be shown

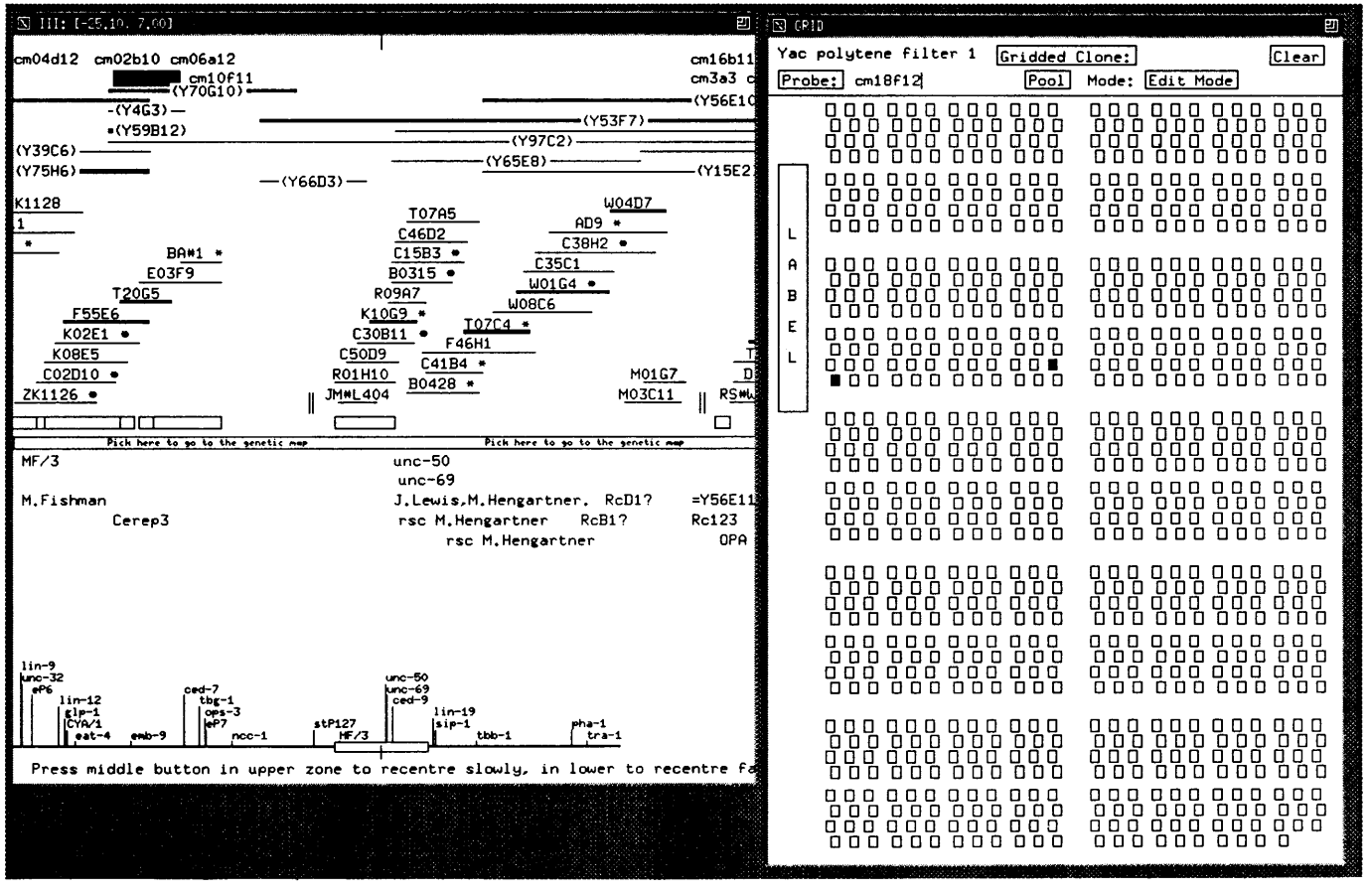

Figure 2: Physical map window and hybridisation grid window.

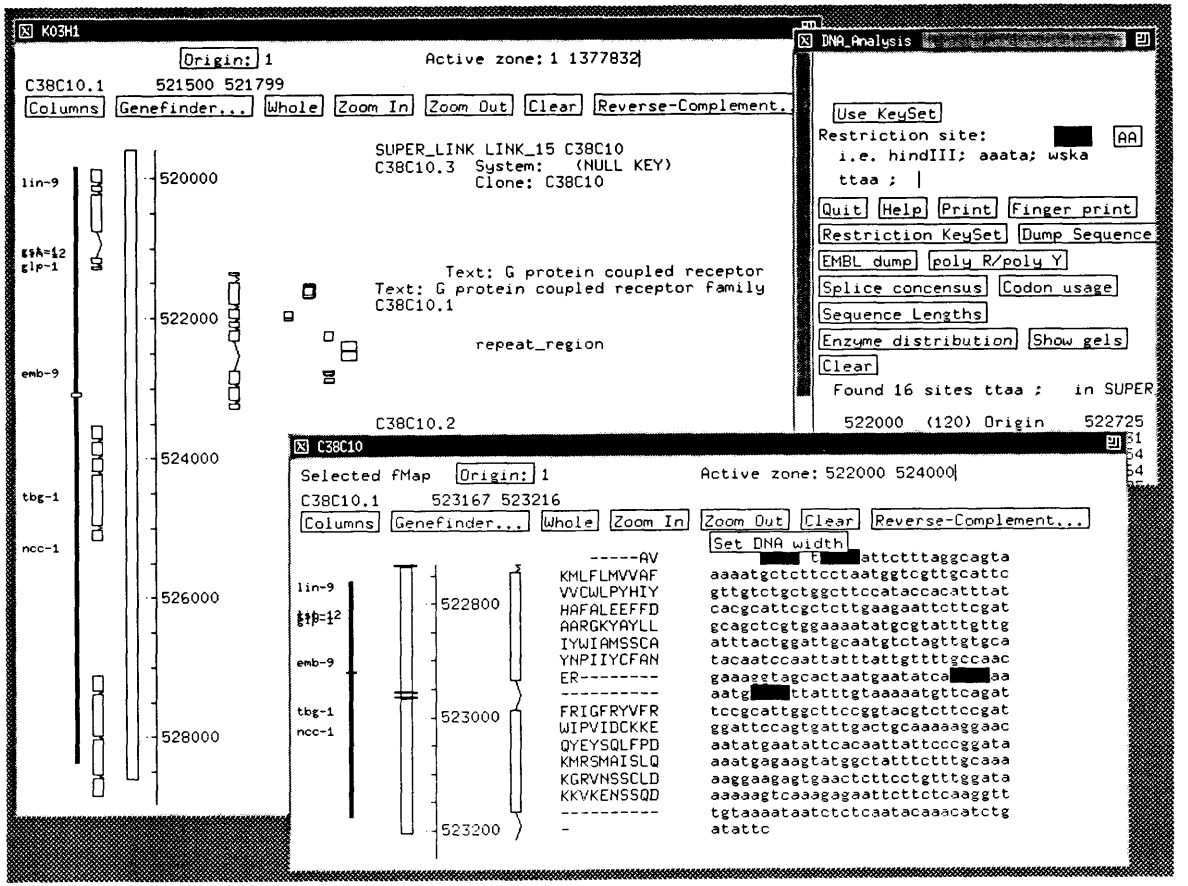

Figure 3: Sequence windows.

with at many different scales, with simplified results on a megabase scale at one extreme, and details of short tandem repeats at the other. Selection of exactly which options are shown is controlled by a separate menu.
As well as displaying annotations and precalculated information, the sequence window supports several types of calculation. In particular there are a range of facilities derived from the Genefinder program (Green and Hillier, personal communication) for predicting gene structures in genomic DNA sequence based on likelihood predictions of splice sites and codon usage. These are in fact used to annotate the nematode genomic DNA sequence before submission to EMBL. There are also restriction site detection tools, and tools for extracting subsequences and translations of predicted genes.

ORGANISATION OF DATA

Definition of the schema

ACEDB is an object-oriented system. This type of organisation of data is much more intuitive than one based on relational tables. Because of this it permits more direct input from working biologists on the construction and refinement of the schema, or data structure.
Each object is represented by a unique identifier, its name, which is followed by an ensemble of attributes organised into a tree. The nodes at branchpoints of the tree are all named. The branches typically terminate in pointers to other objects, or data, which are numerical values, character strings. A bare branch ending just in the named branchpoint can be used to indicate presence of a binary property. There is also the possibility of constructed subobjects, similar to expanding a leaf node in place recursively into a full object with its own branches, rather than maintaining merely a pointer to an external object. Arbitrary text comments can be attached freely at any point in the tree.
The objects are allocated to classes. Each class has a model, specifying the maximal extension of the branches, and the types of data or classes of pointer permitted at each position. Individual objects, which are instances of the class, in general only have a part of the branching pattern permitted by the model. This approach gives a triple advantage:

a) Poorly studied objects, which are by far the most numerous, take little space on disk and in memory, which strongly increases the speed and efficiency of the program.

b) If one wants to extend the schema, which we do frequently, all that needs to be done is to add another branch to the model. Of course none of the existing objects contain this branch, but they remain valid because there is no requirement for objects to contain any particular branch of the model.

c) The ability to add personal comments that are ignored by searching algorithms allows flexible annotation of data sources, reservations etc. without affecting internal search procedures.

Here is a part of the definition of the class Gene, and an example of a gene:

```
?Gene    Reference_Allele ?Allele
         Molecular_information   Clone    ?Clone XREF Gene
                                 Sequence ?Sequence XREF Gene
         Map      Physical pMap UNIQUE ?Contig XREF Gene UNIQUE Int
                           Autopos
                  Genetic  gMap ?Chromosome XREF Gene UNIQUE Float UNIQUE Float
                  Mapping_data    2Point ?2_point_data
                                  3Point ?3_point_data
         Location ?Laboratory #Lab_Location
?Lab_Location    Freezer         Text
                 LiquidN2        Text
```

```
ced-4   Reference_Allele n1162
        Molecular_information   Clone   MT#JAL1
                                Sequence ?Sequence XREF Gene
        Map     Genetic  gMap III -2.7
                Mapping_data    2Point "ced-4 unc-32/+ +"
        Location Cambridge Freezer A6
```

Note that each object belongs to just one class. We have deliberately chosen to avoid multiple inheritance. This concept is at the same time notoriously difficult to implement efficiently and very difficult to use, because the inheritance graph easily becomes encumbered with potential conflicts amongst super-classes.

Our alternative to multiple inheritance is to restrict the number of classes, but allow a wider variety of objects within a class. In this system, it is possible that two objects in the same class have few or no branches in common. For example consider two genes, the first studied by classical genetics and uncloned, the second cloned by similarity to a protein in another organism. These objects could be considered as archetypes of two subclasses of the class gene. But such simple cases are relatively rare, a third gene could have data for some fields of one type, and some of the other, and one is rapidly led to a combinatorial explosion in the number of classes. Our approach lets us capture without difficulty all the intermediate cases, and we only need around fifty classes to hold nearly a hundred thousand heterogeneous objects.

As well as the classes of tree objects described above, also denoted type B classes, we have type A classes, which contain general arrays of data, and which allow more rigid but more efficient storage of data such as DNA sequences.

The schema itself is stored in objects within the database, allowing a simplified startup procedure and dynamic editing of some features.

Ace files: an ascii edit language

Although data are stored internally in a binary form of the trees discussed above, they are normally entered via simple ascii files known as ace files.

Each paragraph in these files corresponds to one object, and must be separated from the next by one or more blank lines. The first line indicates the class and name of the object to be created or edited. Following lines start with the name of a branch node, followed by numerical or text data, or names of other objects to be pointed to. They are interpreted according to the model. Keywords such as -D or -R specify actions to be taken, with the default action being to add the data into the database. As in C++, // indicates a comment in the file.

Example:

```
// First let us define a sequence:
Sequence ACT3
Title ``C. elegans actin gene (3)''
Library EMBL CEACT3 X16798

// next the corresponding DNA (A class with special reader)
DNA ACT3
aagagagacatcctcccgctcccttcccacacccacttgctcttttctat
tgaccacacattatgaagataaccatgttactaatcaaattcgtgttctt
ttccaatttctttttc

// here we change the name zk643 (if it exists)
-R Sequence zk643 ZK643            // R for ``rename''
// here we change one of the authors of the paper [wbg101]
Paper Nurture:7:234-242
-D Author ``Kimble JE''            // deletion of Kimble
Author ``Ahringer JA''             // addition of Ahringer
```

It is specifically because the objects have a public unique ascii identifier, the doublet [class:name], that these edit commands are well-defined and can refer to precise objects. If the object is not known yet, it is created, else it is modified. If a delete or rename operation finds nothing to delete or rename it moves on silently. If an instruction makes no sense according to the model, for example by referring to an unknown branch point, the user is warned and the paragraph is skipped. Together these properties also allow repeated reading of the same file without changing the database contents, and transfer of information between databases that may not match exactly, something which is very hard with traditional database systems. Indeed they even allow transfer of commonly meaningful data between systems whose schemas differ.

As well as reading in data, we can also export a set of objects in ACE file form. An external program, acediff, takes as input two such ace files, and generates a third that would have the effect of transforming a database containing data as in the first file into one containing data as in the second file. This program can be used to generate update files for remote copies of a central database, and in fact this is the procedure we use to distribute the nematode genomic database. There are also facilities within ACEDB for certain types of specialised data output, such as DNA in FASTA format.

THE DATABASE KERNEL

We wanted ACEDB to be portable and efficient. The system is therefore built using the standards of C, Unix, X11 and PostScript. Because we ended up defining a new type of database system we had to write our own database management system. We also wrote our own graphical library on top of X, which can be reimplemented using other underlying graphical systems (e.g. Apple Toolbox), and a number of macro-based extensions to C that seemed useful to us. ACEDB is therefore not tied to any particular machine, nor operating system, nor even (after a little work) windowing system. It contains an internal help system, and a crash recovery system (when possible). Some of these functions are described in this technical section.

Disk storage

The conceptual unit of transfer between disk and memory is the object. Since the objects are trees or arrays of arbitrary size, we wrote a relatively complex module to pack them into and out of fixed size blocks (one or two kilobytes). The many small objects are brought together in groups that each fill one block; large ones are split over several blocks. The system is speeded up by two levels of internal cache, the first acting on the fixed size blocks, the second on developed trees. These both work as last referenced/last out queues.

Since the class of an object is known at all levels, it is easy to selectively optimise the storage of certain classes, as we do for example with DNA.

Within any particular session, all modified objects are rewritten to new disk locations, which allows us to store multiple versions of an object, and recover from crashes by going back to the last verified save state of the database.

Indexing system

Each object is identified externally by the ASCII doublet [class:name]. Internally it is represented by a unique 32 bit key, 8 bits being used for the class, and 24 for the location within the class. Linking these are hash tables that map the known names of each class to keys. For each key there are then a set of indices containing the disk address of the object, cache addresses if it is in memory, flags indicating its edit state and other properties, a pointer back to its name, etc. There are separate index and hash tables for each class. These are loaded into memory as needed, and take up about 30 bytes per object.

Only one user at a time is permitted write access. The set of changes made until write access is given up constitutes a session. When a session is saved, first the changed objects are flushed to disk, then the indices and hash tables for any altered classes are written (as type A class objects, and hence also to new disk locations). Finally a pointer in the superblock is changed to point to the new index information. Once this is done the system will start up with the new indices. Any crash before this point will leave the system so it starts up by retrieving the old indices, and hence the old objects from before the aborted session.

Data access library

All simple access to data within the program uses a subroutine library that allows access via the names of terminal branchpoints, rather like with the ace files. There are two steps: the first recalls an object from disk given a key, and returns a handle; the second uses this handle to recover data. Since any data may be missing, these routines all return a boolean value indicating success or failure. As an example, to recover the date of a paper, one might write (in C code):

```
void paperDate (KEY paper)
{ int year; OBJ obj;
 if ((obj = bsCreate(paper)) && bsGetData(obj,_Year,&year))
   printf (``Paper %s was published in year %d\n", name(key), year) ;
 bsDestroy(obj) ;}
```

Query language and Keysets

As well as accessing data via the direct subroutine package, there is a powerful and general query language that allows higher level manipulation of sets of objects, which are known as keysets. The basic operations in the language are (1) perform filtering operations on a set of objects, either on the basis of their names or the data they contain, and (2) to follow pointers to retrieve other objects. These operations can be combined using boolean operations into complex query sentences.

The resulting keysets can be used in in various ways: single items can be looked at interactively, the whole set can be a starting point for further queries, it can be dumped out as ascii ace file (see above), or it can be saved in the database with a user-specified name. Boolean set operations can also be used to combine sets. An important feature is that sets can contain objects from many classes. One example of how this is used is via another query operation, "text search". This performs a search on all text stored in the database, and returns a list of all objects that either have names matching the search string, or contain text that matches it. For example a search for "muscle" might return genes with muscle phenotypes, papers with "muscle" in the title, sequences of muscle proteins, etc.

The query package is available both to the user, via an interactive interface that allows saving, recovery and reuse of queries, and to the programmer via a library of subroutines. In fact the main control window is implemented by setting up a limited set of straightforward queries.

There is another facility for general data presentation based on the query package, called the "Table Maker". This allows the user to construct tables of displayed information in a similar way to using a spreadsheet. The difference is that, in the table maker, new columns are derived from previous columns by queries, not by calculations. Once again, the instructions for defining a table can be built up interactively, and stored in a file once they are correct.

Textace and the server/client architecture

All the discussion so far has concerned the graphical version of ACEDB that most users see. However there is also a text version of the program, textace, that can be run from the Unix command line. This basically contains the kernel with a simple command parsing interface.

Although it can be useful for extracting data over a serial line, the main purpose of this version is to enable a client/server architecture for ACEDB.

The way this is done is for the server to contain a full copy of the database, and for the client to start with an empty database. When a query is generated by the client, the server resolves it and sends the result back to the client as an ace file, which is parsed into the client database in the normal fashion. In fact the server is acting in exactly the same fashion as an independent textace program, except that it is connected to the client by a pair of sockets, rather than by the standard I/O. This type of structure is only possible because ACEDB allows meaningful data transfer between non-identical databases, via ace files. The client database can either be allowed to accumulate during the session, acting as a local cache, or can be restricted so that all calls for data are resolved by passing them back to the server. Of course the former can become much more efficient, while susceptible to data becoming stale if it is edited by another process. It is clear that when editing data, the objects must be retrieved from the server and locked there, rather than updated based on a local copy.

CONCLUSION

ACEDB is publicly available by anonymous ftp, both in binary form and as source files. There are three primary ftp sites: cele.mrc-lmb.cam.ac.uk (131.111.84.1) in England, directory pub/acedb; lirmm.lirmm.fr (193.49.104.10) in France, directory genome/acedb ncbi.nlm.nih.gov (130.14.20.1) in the USA, directory repository/acedb. In each case the file NOTES gives further instructions on retrieving the program. The data for the current version of the C. elegans database is available in the same directories.

Although we make the source code available (under a licence restricting commercial exploitation), and we encourage development of new specific application code, we hope that the community of groups using ACEDB can keep to a single database kernel. This can be achieved by establishing good contact between groups that are doing development work, and folding kernel changes back into the official release version described above. With this policy, we believe that ACEDB can continue to support a growing community of genome database providers, covering many different genome projects.

REFRENCES

1. "Software for genome mapping by fingerprinting techniques", J. Sulston, F. Mallett, R. Staden, R. Durbin, T. Horsnell and A. Coulson, Comput. Applic. Biosci. 4, 125-132 (1988).